Voldemort NOSQL Database

Swetha Prabha Chaganti
Department of Computer Science
University of Bridgeport, Bridgeport, CT

Abstract

Distributed storage is the backbone of many of the largest scale web sites. Traditionally these systems have been the province of web giants like Google, Yahoo, and Amazon; but a recent surge of open source systems have begun to make this available to a much broader user base. This includes the design and implementation of Project Voldemort, an open source distributed storage system originally developed at LinkedIn.
Voldemort handles a big chunk of traffic at Linkedin serving thousands of requests per second over terabytes of data.
Voldemort is a distributed data store that is designed as a key-value store used by LinkedIn for high-scalability storage.
Voldemort is still under development. It is neither an object database, nor a relational database. It does not try to satisfy arbitrary relations and the ACID properties, but rather is a big, distributed, fault-tolerant, persistent hash table. A 2012 study comparing systems for storing APM monitoring data reported that Voldemort, Cassandra, and HBase offered linear scalability in most cases, with Voldemort having the lowest latency and Cassandra having the highest throughput.

Motivations

Voldemort is basically just a big, distributed, persistent, fault-tolerant hash table. For applications that can use an O/R mapper like active-record or hibernate this will provide horizontal scalability and much higher availability but at great loss of convenience. For large applications under internet-type scalability pressure, a system may likely consist of a number of functionally partitioned services or APIs, which may manage storage resources across multiple data centers using storage systems which may themselves be horizontally partitioned. For applications in this space, arbitrary in-database joins are already impossible since all the data is not available in any single database. A typical pattern is to introduce a caching layer which will require hashtable semantics anyway.

Design and Implementation

To enable high performance and availability we allow only very simple key-value data access. Both keys and values can be complex compound objects including lists or maps, but none-the-less the only supported queries are effectively the following:

value = store.get(key)
store.put(key, value)
store.delete(key)

Step 1: Download and build the code
Run these commands to get the latest and greatest version of Voldemort:

```
git clone https://github.com/voldemort/voldemort
cd voldemort
./gradlew jar
```

Step 2: Start single node cluster
>bin/voldemort-server.sh config/single_node_cluster > /tmp/voldemort.log &

Step 3: Start commandline test client and do some operations

```
> bin/voldemort-shell.sh test tcp://localhost:6666
Established connection to test via tcp://localhost:6666
> put "hello" "world"
> get "hello"
version(0:1): "world"
> delete "hello"
> get "hello"
null
> help
...
> exit
k k thx bye.
```

Characteristics

1. Voldemort uses in-memory caching to eliminate a separate caching tier.
2. Voldemort reads and writes scale horizontally.
3. Automatic replication of data over multiple servers.
4. Automatic partitioning of data, so each server contains only a subset of the total data.
5. Transparent handling of server failure.
6. Pluggable serialization support.
7. Versioning of data items to maximize data integrity.
8. No central point of failure as each node is independent of other nodes.

Result

```
bootstrapTime=1458687151442
context=
deploymentPath=/home/training/voldemort
localHostName=localhost.localdomain
sequence=0
storeName=test
updateTime=1458687150572
releaseVersion=1.10.13
clusterMetadataVersion=0
max_connections=50
max_total_connections=500
connection_timeout_ms=500
socket_timeout_ms=5000
routing_timeout_ms=5000
client_zone_id=-1
failuredetector_implementation=voldemort.cluster.failuredetector.ThresholdFailureDetector
failuredetector_threshold=95
failuredetector_threshold_count_minimum=30
failuredetector_threshold_interval=300000
failuredetector_threshold_async_recovery_interval=10000
 [main]
[18:52:31,733 voldemort.client.AbstractStoreClientFactory] INFO Client zone-id [-1] Attempting to get raw store [voldsys$_met
adata_version_persistence]  [main]
[18:52:31,741 voldemort.client.AbstractStoreClientFactory] INFO Client zone-id [-1] Attempting to get raw store [voldsys$_sto
re_quotas]  [main]
Established connection to test via [tcp://localhost:6666]
> put "hello" "world"
> get "hello"
version(0:1) ts:1458687170710: "world"
> delete "hello"
> get "hello"
null
> exit
bye.
```

Conclusion

Voldemort is a distributed key-value storage system. Voldemort is used at Linkedin. The result was that writes are faster in Cassandra. Reads are faster in Voldemort, MongoDB than Cassandra. These results were for a single-node cluster.

References

1. http://www.project-voldemort.com/Voldemort/
2. https://en.wikipedia.org/wiki/Voldemort_(distributed_data_store)
3. https://www.linkedin.com/in/jaykreps
4. http://blog.search-computing.net/2010/05/project-voldemort-a-distributed-fast-and-reliable-key-value-store/
5. https://engineering.linkedin.com/espresso/introducing-espresso-linkedins-hot-new-distributed-document-store